Name = G Hemant Lakshman Email: hemantlakshmangontu@gmail.com
Reg.no = 11812852 GitHub repository link: https://github.com/Ghlakshman/OS_CA_3

1. **DESCRIPTION:**

The problem assigned to me was to create a “Program ID Manager” which would keep track of all the free program ID and then assign them to the process or threads which request the program ID.

The manager also keeps track of the program Id’s which have been released by the current threads or processes. Now second part of the problem was to create n threads say 50 which would request for program ID and sleep for a random time and then release it.

Now the challenge behind this problem statement was choosing an appropriate data structure for allocation and deallocation of the process ID. Now array was the best shot here because I allow random access to memory while the threads release the PID’s. In terms of Operating systems there should be no ambiguity between the PID’s allocated to the threads.

I had taken an array of size ”900” which would work as a bit map and the index which is assigned as PID, will change the value at that index to “1” which would show the use of that PID by a thread or a process.

2.**ALGORITHM:**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
int pid_array[900],i;//global variables which are used by the threads in common.

int allocate_map()
{
int result;                                  //constant complexity
for( i=0;i<900;i++)//allocating data structure      //900 constant time complexity
{
pid_array[i]=0;          //constant time comlexity
```

```
}

for( i=0;i<900;i++)//checking if memory is allocated properly or not   //900 constant time complexity
 {
if(pid_array[i]!=0) result = 0;
else result = 1;
}
return result;
}

int allocate() //manages the pid allocation
{
int p;
for( i=0;i<900;i++)//checking for free pid //constant time complexity
{
if(pid_array[i]==0){pid_array[i]=1;p=i;break;}
}
return p+100;
}

void release_pid(int p)
{
p = p-100;
pid_array[p] = 0;
}
```

Name = G Hemant Lakshman
Reg.no = 11812852
Email: hemantlakshmangontu@gmail.com
GitHub repository link: https://github.com/Ghlakshman/OS_CA_3

```
void *test_thread(void *arg)//function to create threads and test the pid_manager.
{
int *s = (int *)arg;
int process_id;
process_id = allocate();
sleep(*s);//taking input from user and make the program sleep for a while
printf("\tThread with Process ID: %d\n",process_id);
release_pid(process_id);
}

int main()
{
printf("\t\t\t\t\t********PROGRAM  ID MANAGER********\n");
int inp,lp=1;
printf("\n");
printf("\n");
printf("\n");

printf("\t1.Allocating Memory\n");
int r = allocate_map();
if(r == 1){printf("\tMEMORY ALLOCATED SUCESSFULLY!!\n");}
else printf("\tMEMORY NOT ALLOCATED PROPERLY\n");

printf("\n");
printf("\n");
printf("\t2.Creating Threads  ( Create 100 threads which would print thier respective pid )\n");
```

```
printf("!!WAIT TILL ALL THE THREADS PRINT THIER PID AND THEN ENTER 0 to TERMINATE THE PROGRAM!! \n");

int sl;

pthread_t newthread;

printf("\n");

printf("\n");

printf("\t**Enter how much time the thread will sleep:\t");


scanf("%d",&sl);

printf("\n");

for(i=0;i<100;i++) pthread_create(&newthread,NULL,test_thread,&sl);


while(lp)//loop hich would prevent termination of prrogram and wait for user response to terminate the program

{                    //depends on how much time user takes to terminate the program (n - linear)

   int t;

   scanf("%d",&t);

   if(t==0) lp=0;


}


}
```

TIME COMPLEXITY:

1.All the normal instruction would take constant time or unit time complexity.

2.The **FOR LOOP's** which are running from 0 to 900 have a constant time complexity because there is no increase in graph with respect to input.

Name = G Hemant Lakshman
Reg.no = 11812852
Email: hemantlakshmangontu@gmail.com
GitHub repository link: https://github.com/Ghlakshman/OS_CA_3

3.The **WHILE LOOP** at the end of the program which prevents premature termination of the program without letting all the threads to print their PID's would increase the time complexity of the program in a linear manner:

The increase in time is directly proportional to the time taken by the user to give input to terminate the program.

The relation here is linear in nature.

4. Which sums up the whole complexity of the program to linear – **"N".**

TIME COMPLEXITY = O(n)

PROBLEM CONSTRAINTS:

There was only one problem constraint over there in the problem

That was:

MIN PID = 100 and MAX PID = 1000

I could have solved the constraints problem by defining them as 100 and 1000 for min and max respectively but this could have led to a wastage of 100 memory blocks[0-99] in the data structure I had taken that is array.

I came up with this idea of declaring my array of size 900 containing index from [0-899].

To respect the constraints, I had just done simple step of adding value 100 while I allocate the PID and while deallocating subtracted 100 to get the correct index of the mapping to free the PID.

CODE SNIPPET:

```
int allocate() //manages the pid allocation

{
```

```
int p;
for( i=0;i<900;i++)//checking for free pid
{
if(pid_array[i]==0){pid_array[i]=1;p=i;break;}
}
```

return p+100; -> this here ensure the PID's are within the constrains

```
}
```

```
void release_pid(int p)
{
```

p = p-100; -> this here ensure the PID's are within the constrains.

```
pid_array[p] = 0;
}
```

PURPOSE OF USE:

Process ID is one the important property of any process these are maintained and stored by the program counters these help the OS to track down the process halt them prioritize them and start them.

All process are accessed with thier process ID only.

Now the manager is very important here we can't let two process having same ID which would lead to ambiguity and then lot of synchronization problems in operating systems

Name = G Hemant Lakshman
Reg.no = 11812852
Email: hemantlakshmangontu@gmail.com
GitHub repository link: https://github.com/Ghlakshman/OS_CA_3

BOUNDARIES:

As mentioned, the boundaries of the code would be the range of the PID's the manager can allocate and the problem with the code is the premature termination.

As a lot of threads are created at a time and all will sleep for a while the program finishes execution of all the commands and then doesn't let the threads to implement their code

To solve that issue I had created a while which would wait for the user to end the program on his own wish by passing the input "0" to terminate the program.

TEST CASES:

1.The program was tested on the basis of the output printed by threads.

Each and every thread prints it current holding PID and leaves it after sleeping for a random amount of time.

As the user the only input we give to the computer is to terminate the program no such test cases arise.

TEST CASE 1:

INPUT: **"0"**

EXPECTED OUTPUT: Terminates the program.

OUTPUT: terminates the program.

RESULT : satisfies the test case.

Name = G Hemant Lakshman
Reg.no = 11812852
Email: hemantlakshmangontu@gmail.com
GitHub repository link: https://github.com/Ghlakshman/OS_CA_3

NO of REVISION/COMMITS made in GITHUB = 5.

GITHUB Account: https://github.com/Ghlakshman.